restart;

numanal[divided_diff] := proc(xx::list,y::list,r::name)
    local N, I, X, Q, J, K, P, PP, C, L, M;

    if nops(xx)<nops(y) then
        ERROR("Both lists must be the same size");
    fi;
    N:=nops(xx)-1;

    for I from 0 to 2*N do
        for J from 0 to N do
            Q[I,J]:=infinity;
        od;
    od;

    for I from 0 to N do
        X[2*I] := xx[I+1];
        Q[2*I,0] := round((10.^7)*y[I+1])/(10.^7);
    od;

    STEP 1

    for J from 1 to N do
        for I from J to 2*N-J by 2 do
            Q[I,J] := round((10.^7)*(Q[I+1,J-1] - Q[I-1,J-1]) / (X[I+J] - X[I-J]))/(10.^7);
        od;
    od;

    STEP 2

    printf(` i    x[i]        y[i]
`);
    printf(` -    ----        ----
`);

    for I from 0 to 2*N do
        if Q[I,0]<>infinity then
            printf(`%2d %7.3f %11.7f`, I/2, X[I], Q[I,0]);
        else
            printf(`                      `);
        fi;
        for J from 1 to min(N,4) do
            if Q[I,J]<>infinity then
                printf(` %11.7f`,Q[I,J]);
            else
                printf(`            `);
            fi;
        od;
        printf(`
`);
    od;
    if N>4 then
        K:=iquo(N,4)-1;
        L:=irem(N,4);
        if K>0 then
            for I from 0 to 2*N do
                if Q[I,0]<>infinity then
                    printf(`%2d %7.3f %11.7f`, I/2, X[I], Q[I,0]);
                else
                    printf(`                      `);
                fi;
                for J from 1 to min(N,4) do
                    if Q[I,J]<>infinity then
                        printf(` %11.7f`,Q[I,J]);
                    else
                        printf(`            `);
                    fi;
                od;
                printf(`
`);
            od;
            for I from 0 to 2*N do
                if Q[I,0]<>infinity then
                    printf(`%2d %7.3f %11.7f`, I/2, X[I], Q[I,0]);
                else
                    printf(`                      `);
                fi;
                for J from 1 to N do
                    if Q[I,J]<>infinity then
                        printf(` %11.7f`,Q[I,J]);
                    else
                        printf(`            `);
                    fi;
                od;
                printf(`
`);
            od;
        fi;
    fi;
end:

print(`
`);
for M from 1 to K do
> for I from 4*M to 2*N-4*M do
> for J from 4*M+1 to 4*(M+1) do
> if Q[I,J]<>infinity then
> printf(` %11.7f`,Q[I,J]);
> else
> printf(` `);
> fi
> od;
> printf(`
`);
> od;
> od;
> fi;
> if L>0 then
> K:=K+1;
> for I from 4*K to 2*N-4*K do
> for J from 4*K+1 to 4*K+L do
> if Q[I,J]<>infinity then
> printf(` %11.7f`,Q[I,J]);
> else
> printf(` `);
> fi
> od;
> printf(`
`);
> od;
> fi;
> fi;

> P:=Q[0,0];
for J from 1 to N do
PP:=1;
for K from 0 to J-1 do
PP:=PP*(x-X[2*K])
od;
PP:=P+Q[J,J]*PP;
od;
printf(`\nThe interpolating polynomial is %a`,P);
PP:=0:
for J from 0 to N do
C[J]:=round((10.^7)*coeff(P,x,J))/(10.^7);
PP:=PP+C[J]*x^J;
od;
r:=PP;
end;

Warning, imaginary unit `I` used as a local variable in procedure numanal[divided_diff]

\texttt{numanal[divided\_diff] := proc(xx::list, y::list, r::name)}
\texttt{local N, I, X, Q, J, K, P, PP, C, L, M;}
\texttt{if nops(xx) <> nops(y) then ERROR("Both lists must be the same size") end if}
\texttt{N := nops(xx) - 1;}
\texttt{for I from 0 to 2*N do for J from 0 to N do Q[I, J] := infinity end do end do}
\texttt{for I from 0 to N do X[2*I] := xx[I + 1]; Q[2*I, 0] := (round(10.^7*y[I + 1]))/(10.^7) end do}
\texttt{for J to N do}
for / from $j$ by 2 to $2*2^N - j$ do 
  $Q[I, J] : = \text{round}((10.^7*(Q[I + 1, J - 1] - Q[I - 1, J - 1]))/(X[I + J] - X[I - J]))/(10.^7)$ 
end do 
end do 
printf(` i x[i] y[i] `); 
printf(` - ---- ---- `); 
for / from 0 to $2*2^N$ do 
  if $Q[I, 0] <> \text{infinity}$ then 
    printf(`%2d %7.3f %11.7f `), 1/2*I, X[I], Q[I, 0] 
  else 
  printf(` `) 
  end if 
for $j$ to $\text{min}(N, 4)$ do 
  if $Q[I, J] <> \text{infinity}$ then 
    printf(`%11.7f `), Q[I, J] 
  else 
  printf(` `) 
  end if 
end do 
printf(`/nonmarkingreturn`); 
end do 
if $4 < N$ then 
  $K := \text{iquo}(N, 4) - 1$; 
  $L := \text{irem}(N, 4)$; 
  if $0 < K$ then 
    for / from $4*M$ to $2*2^N - 4*M$ do 
      for $j$ from $4*M + 1$ to $4*M + 4$ do 
        if $Q[I, J] <> \text{infinity}$ then 
          printf(`%11.7f `), Q[I, J] 
        else 
        printf(` `) 
        end if 
      end do 
    end do 
  end if 
  if $0 < L$ then 
    $K := K + 1$; 
    for / from $4*K$ to $2*2^N - 4*K$ do 
      for $j$ from $4*K + 1$ to $4*K + L$ do 
        if $Q[I, J] <> \text{infinity}$ then 
          printf(`%11.7f `), Q[I, J] 
        else 
        printf(` `) 
        end if 
      end do 
    end do 
  end if 
  $P := Q[0, 0]$; 
  for $j$ to $N$ do 
    $PP := 1$; 
    for $k$ from 0 to $j - 1$ do $PP := PP*(x - X[2*K])$ end do 
    $P := P + Q[J, J]*PP$ 
  end do 
  printf(`The interpolating polynomial is %a; P); 
  $PP := 0$; 
  for $j$ from 0 to $N$ do $C[J] := \text{round}(10.^7*\text{coeff}(P, x, J))/(10.^7)$; $PP := PP + C[J]*x^J$ end do 
  $r := PP$ 
end proc 
> 
> numanal[divided_diff_dir] := proc()
printf(`divided_diff returns the interpolating polynomial.\n\n`);
printf(`The arguments for divided_diff are: \n`);
printf(`(1) the list of x-values\n`);
printf(`(2) the list of f(x) or y-values\n`);
printf(`(3) the variable for returning the polynomial\n`);
printf(`If assigning the result to a variable, have the\n`);
printf(`variable and the 3rd argument the same.\n`);
printf(`If p is the variable for returning the polynomial\n`);
printf(`and has already been given a value,\n`);
printf(`the procedure should be preceded by the statement:\n`);
printf(`p:='p'\n`);
end;

numanal\_divided\_diff := proc()
    printf(`divided_diff returns the interpolating polynomial. \n`);
    printf(`The arguments for divided_diff are: \n`);
    printf(`(1) the list of x-values \n`);
    printf(`(2) the list of f(x) or y-values \n`);
    printf(`(3) the variable for returning the polynomial \n`);
    printf(`If assigning the result to a variable, have the \n`);
    printf(`variable and the 3rd argument the same. \n`);
    printf(`If p is the variable for returning the polynomial \n`);
    printf(`and has already been given a value, \n`);
    printf(`the procedure should be preceded by the statement: \n`);
    printf(`p:='p'\n`)
end proc